A DSL for Specifying Timing Requirements

Arda Goknil
AOSTE Project
UNS-I3S-INRIA
Sophia Antipolis, France
arda.goknil@inria.fr

Marie-Agnès Peraldi-Frati
AOSTE Project
UNS-I3S-INRIA
Sophia Antipolis, France
map@unice.fr

***Abstract*—The engineering of real-time distributed embedded systems becomes more and more complex today due to the amount of new functionalities, constraints applied on these functions and the diversity of hardware supporting software execution and communication. Modeling and analysis of time is a key issue for the correct development of these systems. From an engineering point of view, there is a need of a development process supporting modeling timing requirements at different abstraction levels. In this paper we present a Domain Specific Language (DSL) for specifying timing requirements at the analysis phase of the software development life-cycle. The DSL provides the following features: the modeling of different types of timing requirements, the modeling of symbolic timing expressions, i.e. able to deal with bounded or unset parameters in timing requirements, and the integration of complex concepts of distributed systems such as multi rate and multi clock systems.**

Keywords-Timing Requirements, Domain-Specific Language, Requirements Metamodel, Model Driven Development (MDD), Timing Analysis

I. INTRODUCTION

The engineering of real-time distributed embedded systems becomes more and more complex today due to the amount of new functionalities, constraints applied on these functions (timing, cost reduction, weight, energy saving, etc.) and the diversity of hardware supporting software execution and communication. From an engineering point of view, there is a need of a software development process based on different abstraction levels providing capabilities of modeling functionalities and their constraints. For real-time distributed embedded systems the integration of timing requirements at different levels of abstraction becomes mandatory for a high level analysis of timing behavior early in the software development cycle.

In practice, requirements documents are often textual artifacts with implicit structure. Most of the timing aspects such as duration, period, synchronization, multi form time, arithmetic operators and timing variables are not given explicitly in requirements documents. Supporting these timing aspects is a key point to enable the effective use of analysis tools all along the development process (i.e. from requirements to implementation). In this paper we present a Domain Specific Language (DSL) for specifying timing requirements at the analysis phase of the software development cycle. The DSL provides the following features:

- the modeling of different types of timing requirements with their attributes like *event*, *span*, *jitter*, *minimum* and *maximum duration*
- the modeling of symbolic timing expressions, i.e. able to deal with bounded or unset parameters in timing requirements
- the integration of complex concepts of distributed systems such as multi rate and multi clock systems (software being distributed on different Electronic Control Units - ECUs).

In a recent work [10] some of these features are supported as a part of The Timing Augmented Description Language V2 (TADL2) which is integrated to EAST-ADL [4], an architectural description language. TADL2 focuses on timing design constraints in which you specify the desired behavior of the system related to components, ports and functions in the design architecture. With the DSL, our focus is to specify timing requirements in the requirements analysis level by using similar features. Timing requirements specified in the DSL can be refined into timing constraints in TADL2.

The DSL is based on the requirements metamodel given in [2]. We extended the notion of requirements with types of timing requirements (*delay*, *synchronization*, *repetition* and *periodic*).

Consequently, we propose to extend the metamodel with an explicit notion of time base to support multi form timing. Of course, the creation of relations between different time bases is also a part of the extension of the metamodel. In order to express timing values, the notion of timing expression is introduced. Additionally, all timing expressions are augmented with parameters, which can be free at the highest abstraction level and then progressively defined during software development. As a result, a symbolic timing expression in the DSL is possibly made of a suitable set of arithmetic operators mixing symbolic identifiers (not necessarily set variables) and referring to different time bases. One typical use of this feature is to capture unknown configuration parameters. Inherent to this work is also the study of the allowable ranges for symbolic values that are dictated by a set of requirements. The DSL is illustrated with a Brake-By-Wire (BBW) industrial example.

The paper is organized as follows. In Section II we give the BBW system and the associated timing requirements as a

running example. Section III presents the metamodel for timing requirements. In Section IV, we explain the part of the metamodel for timing expressions. Section V gives the details of the modeling environment for the DSL. Section VI gives the related work. In Section VII, we conclude the paper.

II. EXAMPLE OF A BRAKE-BY-WIRE APPLICATION

We use the requirements of a distributed Brake-By-Wire (BBW) application with anti-lock braking functionality in order to illustrate the DSL. The brake-by-wire application is one of the validator proposed by Volvo Technology in TIMMO-2-USE project [12].

The BBW is composed of two mains functions. First a brake controller reads wheel speed sensors and a brake pedal sensor. The brake controller computes the desired brake torque to be applied to the wheels. In addition to this basic brake controller functionality, a second function ABS (Anti blocking System) adapts the brake force on each wheel if the speed of one wheel is significantly smaller than the estimated vehicle speed. In this case, the brake force is reduced on that wheel until it regains speed that is comparable with the estimated vehicle speed. The ABS takes as input the sensor values on each wheel and the estimated vehicle speed. The following is one of the timing requirements for the BBW system about the delay between the brake pedal activation and the brake actuation.

*TR1: There is a **delay** which is measured from the brake pedal stimulus to the brake response. Here, the activation of the brake pedal is the stimulus and the brake actuation is the response. The delay is bounded with a minimum value of X ms and a maximum value of Y ms.*

The minimum and maximum values of the delay are unset. The timing requirement *TR1* will be refined and associated with functions in the design that cover sensor acquisition, brake controller, ABS and Brake Actuation where the values are supposed be set.

The second requirement is about the period of the wheel sensor acquisition.

*TR2: The acquisition of wheel sensors must be done **periodically** every 10 ms.*

TR2 specifies an event (the acquisition of a wheel sensor) which must occur periodically. The timing value for the wheel sensor acquisition is already set in the analysis level.

TR3: First and last wheel brake actuations must follow each other not more than 5 ms.

Since every wheel has its own brake in the vehicle, there will be four brake actuations when the driver activates the brake pedal. Among these four brake actuations, the last one must follow the first one not more than 5 ms.

The following is a timing requirement about the delay between the brake pedal sensor acquisition and the brake controller activation.

*TR4: There is a **delay** which is measured from the brake pedal sensor acquisition to the activation of the brake controller. The minimum and maximum values of the delay are forty percent of the minimum and maximum values of the delay requirement in TR1.*

The minimum and maximum values of the delay in *TR4* are expressed as percentages of the minimum and maximum values in *TR1*. However, we do not know the exact values in *TR4* since the values in *TR1* are unset.

The wheel sensors can be reset in case of malfunctioning. For the accuracy of the sensor measurements the duration between two resets should not be more than 60 seconds. The following is the timing requirement for the wheel sensor resets.

TR5: The wheel sensors might be reset. The duration between two resets of the same wheel sensor should be less than 60 seconds in worst case.

The hardware platform of the vehicle consists of sensors/actuators and computing parts (see Figure 1 with five electronic control units connected by a communication bus). Each ECU has its own timing reference (time base), which is not necessarily (well) synchronized with the other one and the communication between them is still mainly asynchronous (despite the existence of time triggered buses). Such potential drifts between time bases of computing hardware parts (ECU clocks) or latencies in communication parts (bus, memory access, etc.) could be specified in the requirements analysis.

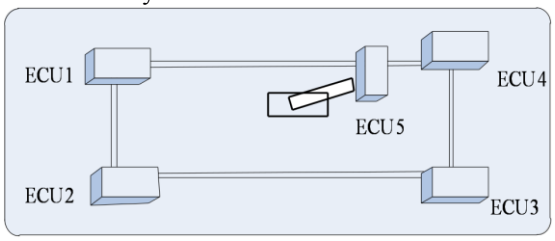

Figure 1. Hardware Platform for the Brake-By-Wire Application

Each brake has its own *ECU* (*ECU1*, *ECU2*, *ECU3* and *ECU4*) and the pedal has *ECU5*. The differences among these ECUs are specified in the following requirements.

TR6: The clock of ECU5 has a drift of 0.02 millisecond for each second compared to the universal time.

TR7: The clock of ECU5 goes 2 times faster than the clock of ECU 1 to 4.

TR6 and *TR7* state that each ECU should have its own timing reference and these timing references have some time differences like drift.

III. REQUIREMENTS METAMODEL FOR TIMING REQUIREMENTS

The DSL is based on the requirements metamodel given in [2]. In [2], we focus on requirements and their relations from a traceability perspective. We aim at improving

requirements relations by assigning relation types and defining their semantics. The semantics of the relations is based on first-order logic (FOL). We use the formalization for consistency checking of the relations and inferencing [2].

Figure 2 gives the metamodel of the DSL which is an extension of the requirements metamodel in [2] for timing requirements. In the DSL metamodel, requirements are captured in a requirements model. A requirements model contains requirements and their relationships. Based on [7], we define a *requirement* as follows.

Definition 1. *Requirement:* A requirement is a description of a system property or properties which need to be fulfilled.

A requirement has a unique *identifier* (ID), *name*, *textual description*, *priority*, *rationale*, and *status*. A system property can be a certain functionality or any quality attribute. In this respect, the *Requirement* entity in the metamodel can be used to model both functional and non-functional requirements.

We identified five types of relations: *requires*, *refines*, *partially refines*, *contains*, and *conflicts* (see [2] for the informal definition of the relations and their formal semantics in first-order logic). Please note that our focus in this paper is not the relations between timing requirements.

In the metamodel, a *timing requirement* is a specialization of a *requirement*. Based on *Definition 1*, we define a *timing requirement* as follows.

Definition 2. *Timing Requirement:* A timing requirement is a description of a property or properties about the desired **timing behavior** of **real-time** systems which needs to be fulfilled by system events.

The DSL supports four types of timing requirements: *Delay Requirement*, *Synchronization Requirement*, *Repetition Requirement*, and *Periodic Requirement*.

Other types of timing requirements identified as timing constraints in the design level could be included in the DSL such as *Burst Requirement*, *Arbitrary Requirement*, *Reaction Requirement* and *Sporadic Requirement*. However, these requirements describe more complicated system properties mainly identified as design constraints. In the DSL, we consider mainly basic requirements which can be identified as system properties (quality attribute) in the analysis level.

The timing behavior described by a timing requirement needs to be fulfilled by system events. We define an *event* as follows.

Definition 3. *Event*: An event stands for all forms of identifiable state changes that are possible to constrain with respect to timing.

For instance, breaking the pedal, acquisition of the wheel sensors and actuating wheel brakes can be identified as state changes in the Brake-By-Wire application.

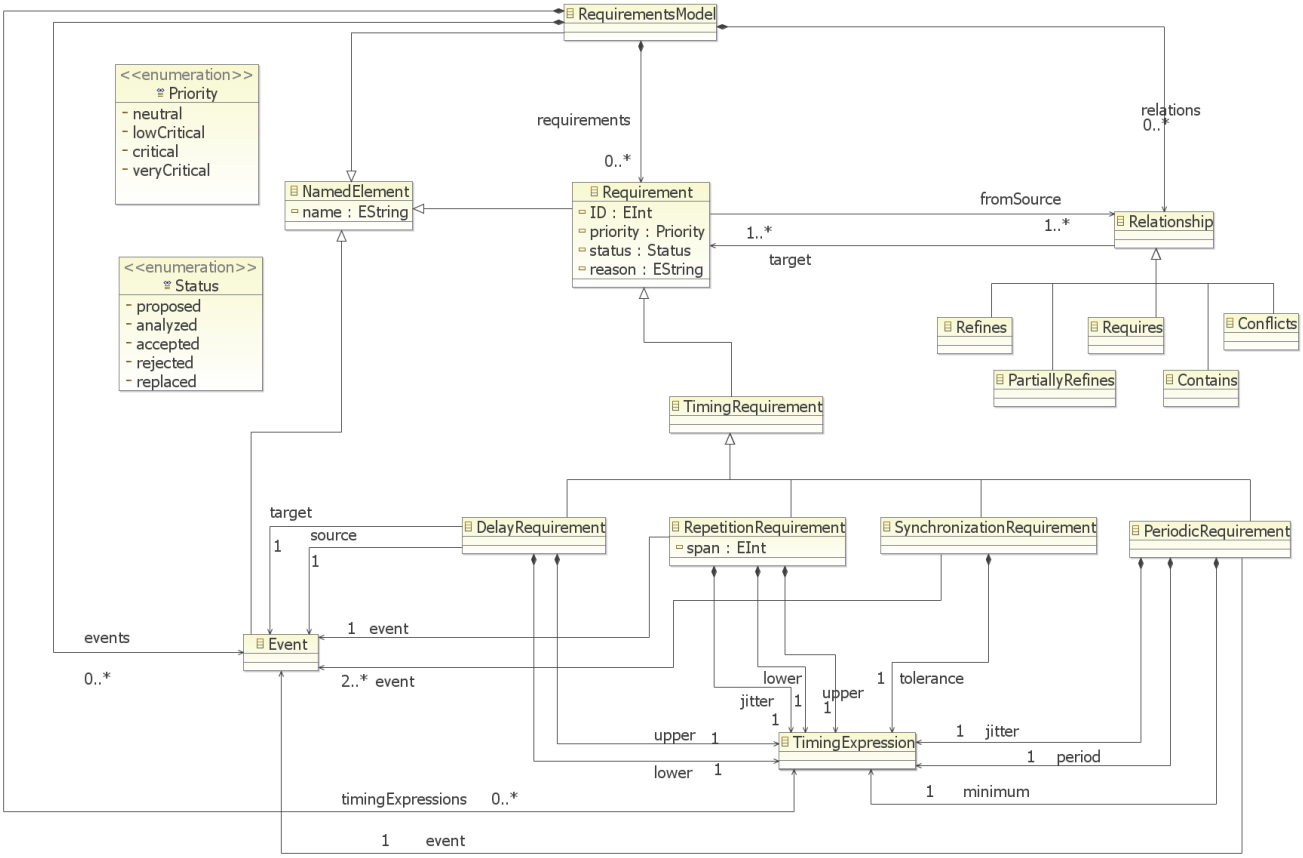

Figure 2. Requirements Metamodel for Timing Requirements

One of the timing requirements is the delay requirement. We describe a *delay* requirement as follows.

Definition 4. *Delay Requirement*: A delay requirement describes how occurrences of an event called *target* are placed relative to each occurrence of an event called *source*.

Every instance of *source* must be matched by an instance of *target*, within a time window starting at *lower* and ending at *upper* time units relative to the source occurrence (see *upper* and *lower* for *DelayRequirement* in Figure 2).

The *TC1* timing requirement in Section II is about the delay between the brake pedal stimulus and the brake response. We provide a textual concrete syntax for the DSL. Listing 1 gives the *TR1* timing requirement with the textual concrete syntax.

```
1   RequirementsModel BBW {
2
3     Event brakePedalActivation { }
4     Event firstWheelBrakeActuation { }
5
6     DelayRequirement dr1 {
7         ID = 1
8         priority neutral
9         status analyzed
10
11        source brakePedalActivation
12        target firstWheelBrakeActuation
13
14        lower = X
15        upper = Y
16
17    } // end of the dr1 delay requirement
18    ....
```

Listing 1 Delay Requirement for Brake Actuation

We set the attributes ID, priority and status as '*1*', '*neutral*' and '*analyzed*' (see line 7-9). The brake pedal activation is defined as an event (see line 3). Since there is one brake for each wheel in a vehicle, we define a single event for one of the wheels (see line 4). Please note that for other three wheels we should specify three similar delay requirements with three similar events. For the source and target events, the delay requirement *dr1* has the attributes *lower and upper* which are equal to the variables *X* and *Y* respectively (see lines 14 and 15). The variables are variable timing expressions which are explained in detailed in Section IV.

Another timing requirement is the *synchronization* requirement which we describe as follows.

Definition 5. *Synchronization Requirement*: A synchronization requirement describes how tightly the occurrences of a group of events follow each other.

There must be a sequence of time windows of width *tolerance*, such that every occurrence of every event in *events* belongs to at least one window, and every window is populated by at least one occurrence of every event (see *tolerance* and *events* for *SynchronizationRequirement* in Figure 2).

The *TR3* timing requirement in Section II is about the maximum tolerated time difference between the first and last wheel brake actuation. Listing 2 gives the *TR3* timing requirement as a synchronization requirement in our DSL.

```
1   Event secondWheelBrakeActuation { }
2   Event thirdWheelBrakeActuation { }
3   Event fourthWheelBrakeActuation { }
4
5   SynchronizationRequirement sr1 {
6       ID = 2
7       priority neutral
8       status analyzed
9
10      events firstWheelBrakeActuation,
11             secondWheelBrakeActuation,
12             thirdWheelBrakeActuation,
13             fourthWheelBrakeActuation
14
15      tolerance = (5.0 ms on universal_time)
16
17  } // end of the sr1 synchronization requirement
18  …
```

Listing 2. Synchronization Requirement for Brake Actuation

The brake actuation is defined for each wheel as an event (see lines 1-3 in Listing 2 and see line 4 in Listing 1). For these events, the synchronization requirement *sr1* has the attribute *tolerance* which is a value timing expression (see line 15). Please note that value timing expressions will be explained in detailed in Section IV.

The third timing requirement is the repetition requirement in which we can specify the distribution of system state changes. We describe a *repetition* requirement as follows.

Definition 6. *Repetition Requirement*: A repetition requirement describes the distribution of a single event.

Every sequence of *span* occurrences of event must have a length of at least *lower* and at most *upper* units (see *lower*, *upper* and *span* for *RepetitionRequirement* in Figure 2).

Listing 3 gives the *TR5* timing requirement as a repetition requirement.

```
1   Event firstWheelSensorReset { }
2
3   RepetitionRequirement rp1 {
4       ID = 3
5       priority neutral
6       status analyzed
7
8       event firstWheelSensorReset
9
10      span = 1
11      lower = (0.0 ms on universal_time)
12      upper = (60.0 second on universal_time)
13      jitter = (0.0 ms on universal_time)
14
15  } // end of the rp1 repetition requirement
16   …
```

Listing 3. Repetition Requirement for Sensor Reset

The *rp1* repetition requirement is given for only the sensor reset of the first wheel. The sensor reset of the first wheel is defined as an event (see line 1). *rp1* defines the

upper limit of two occurrences of the sensor reset as 60 second. Since *TR5* does not state any *lower* or any *jitter*, we assume that the attributes *lower* and *jitter* are zero in Listing 3.

The repetition requirement does not specify any period. Periodic occurrences of an event are specified with a periodic requirement. We describe a *periodic* requirement as follows.

Definition 7. *Periodic Requirement*: A periodic requirement describes an event that occurs periodically.

Listing 4 gives the *TR2* timing requirement as a periodic requirement.

```
1   Event firstWheelSensorAcquisition { }
2
3   PeriodicRequirement pr1 {
4       ID = 4
5       priority neutral
6       status analyzed
7
8       event firstWheelSensorAcquisition
9
10      period = (10.0 ms on universal_time)
11      minimum = (0.0 ms on universal_time)
12      jitter = (0.0 ms on universal_time)
13
14  } // end of the pr1 periodic requirement
15  …
```

Listing 4. Periodic Requirement for Sensor Acquisition

The periodic requirement *pr1* specifies a period which is 10 ms for the acquisition of the first wheel sensor. The *period* attribute is considered as the upper limit for the periodicity. Since *TR2* does not state any *minimum* value or any *jitter*, we assume that the attributes *minimum* and *jitter* are zero.

IV. The Metamodel with Timing Expressions

In this section we introduce the notion of timing expression. One of the timing expressions is *Symbolic Timing Expression (STE)*. A STE is a way to specify parameterized expressions. In the metamodel, *TimingExpression* provides free variables, constants, values and operators to cover the need for symbolic parameterized timing expressions. *TimeBase* together with *Dimension*, *Unit* and *TimeBaseRelation* address the integration of complex concepts of distributed systems such as multi clock systems.

In Section IV.A, we give the part of the metamodel for multi time base extension. Section IV.B depicts the part of the metamodel in order to relate time bases to each other. In Section IV.C, we introduce the timing expressions which are *symbolic timing expression*, *variable timing expression* and *value timing expression*.

A. A Multi Time Base Extension of the Metamodel

RequirementsModel contains *TimeBase* which represents a discrete and totally ordered set of instants (see Figure 3). An instant can be seen as an event occurrence called a "tick". It may represent any repetitive event in a system. Events may refer even to "classical" time dimension or to some evolution of a hardware part (rotation of crankshaft, distance, etc.). The type of *TimeBase* is *Dimension*. *Dimension* has a *kind* that represents the nature of *TimeBase*. Additionally, *Logical* can be used to define a logical time reference. Finally, *other* can be used for very specific applications.

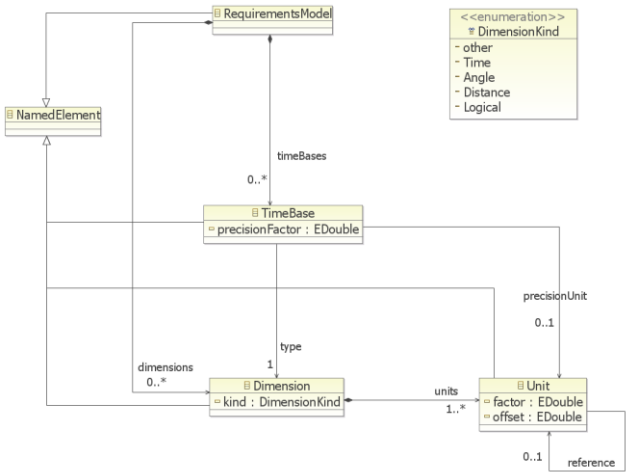

Figure 3. Part of the Metamodel for Time Base and Dimension

```
1   Dimension physicalTime {
2     units {
3         micros{factor 1.0 offset 0.0},
4         ms{factor 1000.0 offset 0.0 reference micros}
5         second{factor 1000000.0 offset 0.0 reference micros}
6     }
7     kind Time
8   }
9
10  TimeBase universal_time {
11    dimension physicalTime
12    precisionFactor 0.1
13    precisionUnit micros
14  }
15
16  TimeBase ecu1 {
17    dimension physicalTime
18    precisionFactor 0.1
19    precisionUnit micros
20  }
21
22  TimeBase ecu5 {
23    dimension physicalTime
24    precisionFactor 0.1
25    precisionUnit micros
26  }
27  …
```

Listing 5. Example of Dimension and TimeBase

Dimension defines the set of units that can be used to express duration measured on a given *TimeBase*. Each *Unit* relates to another unit to enable conversions. The *factor*, *offset* and *reference* attributes in *Unit* are used for such

conversions. Only linear conversions between units of the same dimension are allowed. As a unit conversion example, the unit *second = 1000*millisecond* so *factor = 1000 and offset =0*.

Because *Timebase* is a discrete set of instants, a discretization step is specified with the *precisionFactor* attribute which relies on *precisionUnit*.

Listing 5 shows one *Dimension* and three *TimeBases* (*physicalTime*, *universal_time*, *ecu1* and *ecu5*). For the *physicalTime* dimension, a list of units and attributes for their conversion expression are given.

To avoid the duplication in Listing 5 we do not show time base declarations for *ecu2*, *ecu3*, and *ecu4* (see ECUs in Figure 1).

B. TimeBase Relation

As stated in the timing requirements *TR6* and *TR7* in Section II, there are relations between time bases.

TimeBaseRelation (see Figure 4) is used to give equivalence between different time bases. More precisely, it specifies equality between the *left* and *right* timing expressions.

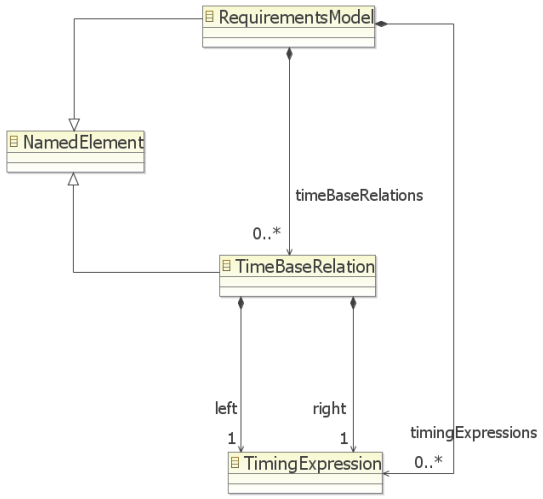

Figure 4. Part of the Metamodel for TimeBase Relations

```
1  TimeBaseRelation tbr1 {
2     (1.0 second on universal_time) = (1.00002 second on ecu5)
3  }
4
5  TimeBaseRelation tbr2 {
6     (1.0 ms on ecu1) = (2.0 ms on ecu5)
7  }
8  …
```

Listing 6. Example of TimeBase Relation

Listing 6 shows the time base relations stated in *TR6* and *TR7* for the BBW example. As stated in *TR6*, *ecu5* has a drift of 0.02 millisecond for each second compared to the universal time. Also, the *ecu5 TimeBase* goes 2 times faster than *TimeBases* of *ecu1* to *4* (see *TR7* in Section II).

The timing requirements *TR6* and *TR7* become the time base relations *tbr1* and *tbr2* in Listing 6. To avoid the duplication, we do not show the time base relations between *ecu2&ecu5*, *ecu3&ecu5* and *ecu4&ecu5*.

C. Timing Expression

TimingExpression stands for all terms that denote time values. There are three types of timing expressions: *ValueTimingExpression*, *VariableTimingExpression* and *SymbolicTimingExpression* (see Figure 5).

The DSL is a declarative language. Therefore, we have only free variables, constants and values. *VariableTimingExpression* stands for free variables and constants. In *SymbolicTimingExpression*, the language integrates basic arithmetic and relation operators such as '*addition'*, '*subtraction'*, '*multiplication'*, '*greater than'*, and '*less than'* associated with timing values.

```
1    Event brakePedalSensorAcquisition { }
2    Event brakeController { }
3
4    DelayRequirement dr2 {
5        ID = 5
6        priority neutral
7        status critical
8
9        source brakePedalSensorAcquisition
10       target brakeController
11
12       lower = Z
13       upper = T
14
15   } // end of the dr2 delay requirement
16
17   var X ms on universal_time    // Variable Timing Expression
18   var Y ms on universal_time    // Variable Timing Expression
19   var Z ms on universal_time    // Variable Timing Expression
20   var T ms on universal_time    // Variable Timing Expression
21
22   {(X < Y)}             // Symbolic Timing Expression (STE)
23   {(Z < T)}             // STE
24   {(Z := 0.40* X)}      // STE
25   {(T := 0.40* Y)}      // STE
26
27 } // end of the BBW requirements model
```

Listing 7. Example Timing Expressions

There are some constraints for the metamodel which can be written in Object Constraint Language (OCL) [15].

- The left hand side of *TimeBaseRelation* cannot be *SymbolicTimingExpression* with *Operator*. It can only be *VariableTimingExpression* or *ValueTimingExpression* with *Unit* and *TimeBase*.
- The right hand side of *TimeBaseRelation* cannot be *SymbolicTimingExpression* with a relation operator such as *Assignment* or *LessThan*. For instances, the following time base relation is not allowed: {(1 degree on crk_angle) = (X < (5 ms on universal_time)}. On the other hand, it is possible to have the following time base relation: {(1 degree on crk_angle) = (X + (5 ms on universal_time))}.

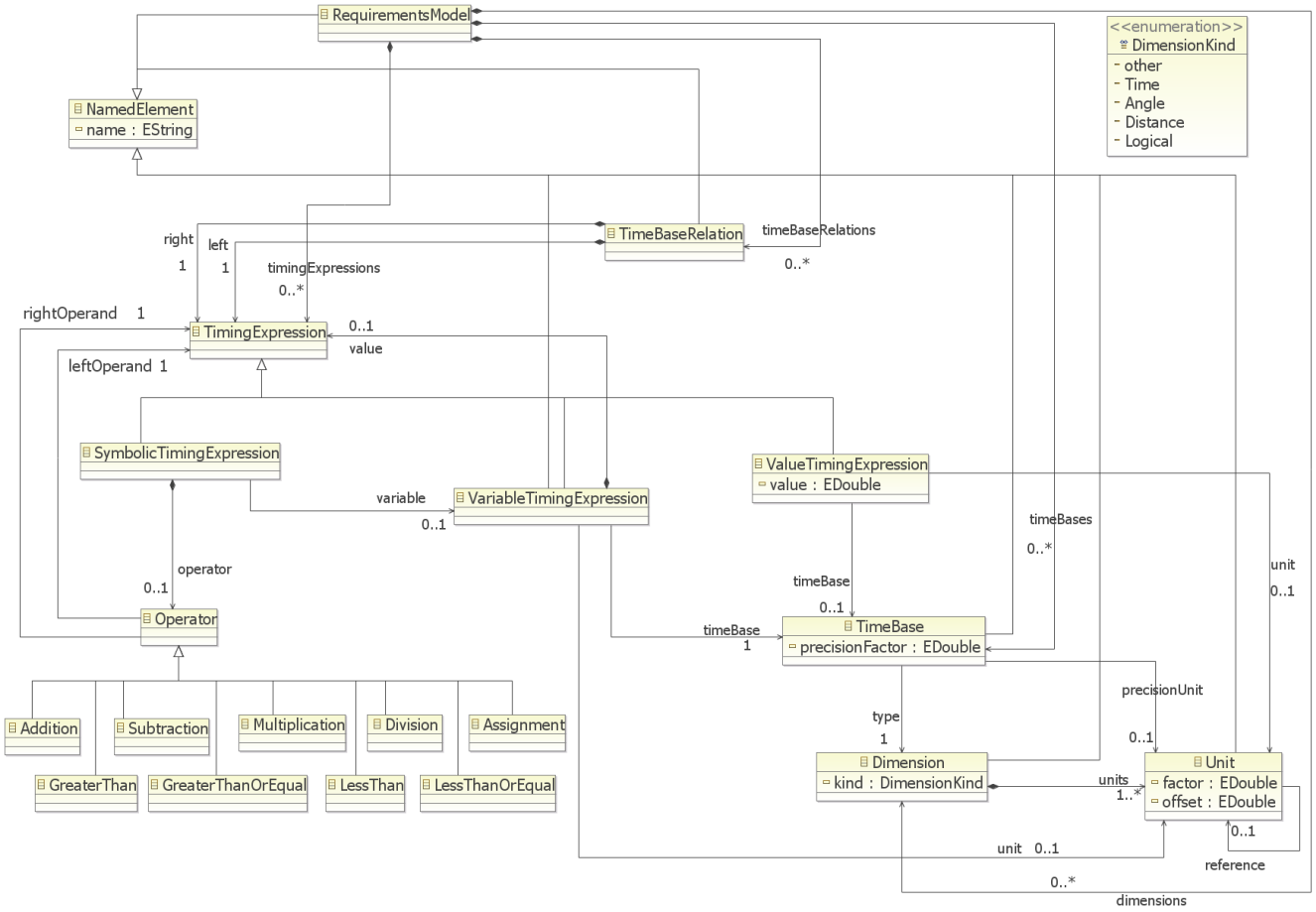

Figure 5. Part of the Requirements Metamodel for Timing Expressions

- The arithmetic operators cannot have right/left operands which are *SymbolicTimingExpression* containing any relation operator. For instance, the following symbolic timing expression is not valid: {((X < (5 ms on universal_time)) + Y)}.

Please note that *TimeBaseRelation* is different than the relation operator *Assignment*. Since we have only free variables and constants, the *Assignment* operator can be used only once for a variable in the left operand. The variable becomes a constant.

Listing 7 extends the *BBW* requirements model with examples of timing expressions. The *var* keyword is used for defining both free variables and constants. Free variables are useful for characterizing parameters in timing expressions and for referring to already existing timing expressions. *X* and *Y* are defined as free variables (see lines 17 and 18) and they are used in the *dr1* delay requirement (see Listing 1). We have two other free variables (*Z* and *T*) used for the lower and upper values of the *dr2* delay requirement. The upper and lower values of *dr2* are forty percent of the upper and lower values of *dr1* respectively (see the timing requirements *TR1* and *TR4* in Section II). The equations for the upper and lower values are given as symbolic timing expressions (see lines 24 and 25).

STE allows comparison of variables. For instance, *X* which stands for the lower value of *dr1* should be lower than *Y* which stands for the upper value of the same requirement (see line 22). Please note that different time bases can be used for different variables in the same relation or arithmetic operation.

V. Modeling Environment for The DSL

We have an editor for the DSL that supports textual concrete syntax. The metamodel is implemented with *ecore* [6] in Eclipse Modeling Framework (EMF). Textual concrete syntax is generated by using *Xtext* [16] which is a framework/tool for development of textual domain specific languages.

VI. Related Work

Most of the approaches in the literature consider timing requirements as constraints in the design level. EAST-ADL [4] and AUTOSAR [5] support timing design constraints for automotive domain.

AUTOSAR [3] [5] is able to express requirements based on a unique and implicit time base. Timing constraints

cannot be parameterized so that they can only be specified later in the development process. They are not amenable to cover complex arithmetic timing expressions.

EAST-ADL [4] and AUTOSAR [5] have only two implicit time bases from two dimensions and cannot express any other time bases (distance, temperature, etc.). AUTOSAR allows modeling units of different nature (ms, s, °, etc.). The relation between these units is a multiplication factor that should be expressed for each timing expression.

In a recent work [10] we propose The Timing Augmented Description Language V2 (TADL2) which supports most of the features provided in the DSL. However, TADL2 focuses on timing design constraints in which you specify the desired behavior of the system related to components, ports and functions in the architecture. It is integrated to EAST-ADL.

Klein and Giese [9] present Timed Story Scenario Diagrams (TSSD), a visual notation for scenario specifications that takes structural system properties into account. TSSD allows designers to specify structural and temporal properties in a comprehensible manner. It provides conditional timed scenarios describing the partial order of specific structural design configurations. It is possible to specify *Time Constraints* which allow setting *lower* and *upper* bounds for delays in these scenarios. Zschaler [11] presents QML/CS, a specification language that is used to model non-functional properties of components and component-based software systems including response time. Alfonso et.al. [1] present VTS, a visual language to define complex event-based requirements like freshness, bounded response, and event correlation. In VTS, an event does not have to refer to a structural design entity. Therefore, in VTS, it is possible to describe the requirements in the analysis level independent from the design of the system. On the other hand, VTS does not support different time units coded as time bases and relations between these time units. Aagedal [14] presents a general modelling language for Quality of Service (QoS). The presented modelling language is based on enterprise architecture modeling. It uses a time model where different clocks can be defined. These clocks are related to the chosen standard clock with skew, drift and offset. They are indirectly related to each other via the chosen standard clock.

VII. CONCLUSION

This paper presents a domain specific language for specifying timing requirements of real-time embedded systems in the requirements analysis level. Types of timing requirements, time bases, time base relations and parameterized timing expressions are parts of the DSL. The language is illustrated with a BBW example.

The DSL currently supports four types of timing requirements (Delay Requirement, Synchronization Requirement, Repetition Requirement, and Periodic Requirement). It is not an arbitrary selection of timing requirements for which we have provided language constructs. In the DSL, we consider mainly basic requirements which can be identified as system properties (quality attribute) in the analysis level. Other types of timing requirements could be included in the DSL such as Burst Requirement and Arbitrary Requirement. These requirements describe more complicated system properties mainly identified as design constraints.

The DSL allows the requirements engineers to specify timing requirements in a structured way. The important aspects of timing requirements such as time base, dimension, equations and variables can be modeled explicitly. The DSL avoids ambiguity and missing details in timing requirements by having an explicit structure of timing requirements including attributes like *jitter*, *period*, and *span*.

In the design level, timing requirements are supported by timing constraint languages like TADL2. For instance, TADL2 provides similar features for specifying timing constraints in the design level. The DSL together with TADL2 support a high level modeling of timing aspects of a system and a refinement of timing requirements through analysis and design levels. Therefore, timing requirements specified in the DSL can be traced to timing constraints in TADL2 for EAST-ADL architecture design models. The impact of any change in timing requirements can be determined for the timing constraints in the design level. Change impact analysis can also be performed in the analysis level for timing requirements. The use of symbolic timing expressions in the DSL allows the requirements engineers to change timing requirements in the requirements model just by performing changes on the relevant parameters.

One potential use of the DSL is composing timing requirements with other non-functional requirements such as power consumption and safety requirements. Similar DSLs can be defined for these non-functional requirements. Safety or power consumption requirements may refer to timing expressions in timing requirements. By composing the DSLs of timing requirements and these non-functional requirements we will have direct references between these requirements as mathematical statements. This will allow us to determine the impact of changes in timing requirements on safety and power consumption requirements by following the mathematical statements.

One future research direction is analyzing timing requirements by using model transformation techniques to go towards simulation and analysis tools. One potential candidate for simulation is the TimeSquare environment [13] and the associated language CCSL [8] which support multi clock system specification. To analyze the consistency of timing requirements by using these simulation and analysis tools we need additional constructs mainly about relations between events in the DSL. We plan to introduce *event chains* to create relations between events.

ACKNOWLEDGMENT

This paper is based on the TIMMO-2-USE project in the framework of the ITEA2, EUREKA cluster program Σ! 3674. The work has been funded by The French Ministry for Industry and Finances, the German Ministry for Education and Research (BMBF) under the funding ID 01IS10034. The responsibility for the content rests with the authors.

REFERENCES

[1] A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero. Visual Timed Event Scenarios. ICSE 2004, Scotland, UK, 2004.

[2] A. Goknil, I. Kurtev, K. van den Berg, and J. W. Veldhuis. Semantics of Trace Relations in Requirements Models for Consistency Checking and Inferencing. Software and System Modeling. 10(1): 31-54, 2011.

[3] AUTOSAR AUTomotive Open System Architecture. http://www.autosar.org

[4] EAST-ADL Language Specification, , http://www.atesst.org/home/liblocal/docs/ATESST2_D4.1.1_EAST-ADL2-Specification_2010-06-02.pdf

[5] AUTOSAR Specification of Timing Extensions, 1.1.0, AUTOSAR Release 4.0.2, 2010-11-03, AUTOSAR Development Cooperation.

[6] Eclipse Modeling Framework (EMF). http://www.eclipse.org/modeling/emf/

[7] Guide to Software Engineering Body of Knowledge. IEEE Computer Society, Colorado. http://www.swebok.org/. Accessed 19 October 2009

[8] F. Mallet, C. André, and R. de Simone. CCSL: Specifying Clock Constraints with UML/Marte. ISSE, 4(3):309–314, 2008.

[9] F. Klein, and H. Giese. Joint Structural and Temporal Property Specification using Timed Story Scenario Diagrams. FASE 2007, LNCS 4422, pp. 185-199, 2007.

[10] M-A. Peraldi-Frati, A. Goknil, J. Deantoni, and J. Nordlander. A Timing Language for Specifying Multi Clock Automotive Systems: The Timing Augmented Description Language. ICECCS 2012, Paris, France.

[11] S. Zschaler. Formal Specification of Non-functional Properties of Component-Based Software Systems. Software and System Modeling. 9(2): 161-201, 2009.

[12] The ITEA TIMMO-2-USE Project. http://timmo-2-use.org/

[13] J. DeAntoni, F. Mallet, and C. André. TimeSquare: on the formal execution of UML and DSL models. Tool session of the 4th Model driven development for distributed real time systems, 2008.

[14] J. Ø. Aagedal. Quality of Service Support in Development of Distributed Systems. PhD Thesis, University of Oslo, 2001.

[15] J. Warmer, and A. Kleppe. The Object Constraint Language: Precise Modeling with UML. Addison-Wesley, 1999.

[16] XText. http://www.eclipse.org/Xtext/